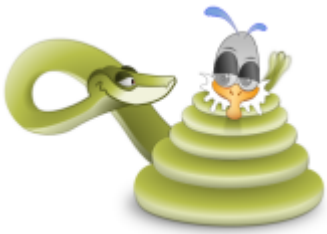
NETWORK PROGRAMMING - SERVER & CLIENT B : FILE TRANSFER



Note

In this chapter, we're going to extend Python Network Programming I - Basic Server / Client A, and try to file transfer from a server to numerous clients. The main purpose is to check the performance of the server from which clients download files.

Local file transfer

Here is the code to send a file from a local server to a local client.

```
# server.py

import socket                   # Import socket module

port = 60000                    # Reserve a port for your service.
s = socket.socket()             # Create a socket object
host = socket.gethostname()     # Get local machine name
s.bind((host, port))            # Bind to the port
s.listen(5)                     # Now wait for client connection.

print 'Server listening....'

while True:
    conn, addr = s.accept()     # Establish connection with client.
    print 'Got connection from', addr
    data = conn.recv(1024)
    print('Server received', repr(data))

    filename='mytext.txt'
    f = open(filename,'rb')
```

```
    l = f.read(1024)
    while (l):
       conn.send(l)
       print('Sent ',repr(l))
       l = f.read(1024)
    f.close()

    print('Done sending')
    conn.send('Thank you for connecting')
    conn.close()


# client.py

import socket                   # Import socket module

s = socket.socket()             # Create a socket object
host = socket.gethostname()     # Get local machine name
port = 60000                    # Reserve a port for your service.

s.connect((host, port))
s.send("Hello server!")

with open('received_file', 'wb') as f:
    print 'file opened'
    while True:
        print('receiving data...')
        data = s.recv(1024)
        print('data=%s', (data))
        if not data:
            break
        # write data to a file
        f.write(data)

f.close()
print('Successfully get the file')
s.close()
print('connection closed')
```

Output on a local server:

```
Server listening....
Got connection from ('192.168.56.10', 62854)
```

```
('Server received', "'Hello server!'")
('Sent ', "'1 1234567890\\n
...
('Sent ', "'4567890\\n105
...
('Sent ', "'300 1234567890\\n'")
Done sending
```

Output on a local client:

```
file opened
receiving data...
data=1 1234567890
2 1234567890
...
103 1234567890
104 123
receiving data...
data=4567890
105 1234567890
106 1234567890
...
299 1234567890

receiving data...
data=300 1234567890
Thank you for connecting
receiving data...
data=
Successfully get the file
connection closed
```

multithread tcp file transfer on localhost

Our server code above can only interact with one client. If we try to connect with a second client, however, it simply won't reply to the new client. To let the server interact

with multiple clients, we need to use multi-threading. Here is the new server script to accept multiple client connections:

```
# server2.py
import socket
from threading import Thread
from SocketServer import ThreadingMixIn

TCP_IP = 'localhost'
TCP_PORT = 9001
BUFFER_SIZE = 1024

class ClientThread(Thread):

    def __init__(self,ip,port,sock):
        Thread.__init__(self)
        self.ip = ip
        self.port = port
        self.sock = sock
        print " New thread started for "+ip+":"+str(port)

    def run(self):
        filename='mytext.txt'
        f = open(filename,'rb')
        while True:
            l = f.read(BUFFER_SIZE)
            while (l):
                self.sock.send(l)
                #print('Sent ',repr(l))
                l = f.read(BUFFER_SIZE)
            if not l:
                f.close()
                self.sock.close()
                break

tcpsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcpsock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
tcpsock.bind((TCP_IP, TCP_PORT))
threads = []


while True:
    tcpsock.listen(5)
    print "Waiting for incoming connections..."
    (conn, (ip,port)) = tcpsock.accept()
```

```
    print 'Got connection from ', (ip,port)
    newthread = ClientThread(ip,port,conn)
    newthread.start()
    threads.append(newthread)

for t in threads:
    t.join()



# client2.py
#!/usr/bin/env python

import socket

TCP_IP = 'localhost'
TCP_PORT = 9001
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
with open('received_file', 'wb') as f:
    print 'file opened'
    while True:
        #print('receiving data...')
        data = s.recv(BUFFER_SIZE)
        print('data=%s', (data))
        if not data:
            f.close()
            print 'file close()'
            break
        # write data to a file
        f.write(data)

print('Successfully get the file')
s.close()
print('connection closed')
```

Below is the output from the server console when we run two clients simultaneously:

```
$ python server2.py
Waiting for incoming connections...
Got connection from  ('127.0.0.1', 55184)
```

```
 New thread started for 127.0.0.1:55184
Waiting for incoming connections...
Got connection from  ('127.0.0.1', 55185)
 New thread started for 127.0.0.1:55185
Waiting for incoming connections...
```

tcp file download from EC2 to local

In the following codes, we made two changes:

1. ip switched to amazon ec2 ip
2. To calculate the time to take download a file, we import **time** module.

```
# server3.py on EC2 instance
import socket
from threading import Thread
from SocketServer import ThreadingMixIn

# TCP_IP = 'localhost'
TCP_IP = socket.gethostbyaddr("your-ec2-public_ip")[0]
TCP_PORT = 60001
BUFFER_SIZE = 1024

print 'TCP_IP=',TCP_IP
print 'TCP_PORT=',TCP_PORT

class ClientThread(Thread):

    def __init__(self,ip,port,sock):
        Thread.__init__(self)
        self.ip = ip
        self.port = port
        self.sock = sock
        print " New thread started for "+ip+":"+str(port)
```

```
    def run(self):
        filename='mytext.txt'
        f = open(filename,'rb')
        while True:
            l = f.read(BUFFER_SIZE)
            while (l):
                self.sock.send(l)
                #print('Sent ',repr(l))
                l = f.read(BUFFER_SIZE)
            if not l:
                f.close()
                self.sock.close()
                break

tcpsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcpsock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
tcpsock.bind((TCP_IP, TCP_PORT))
threads = []

while True:
    tcpsock.listen(5)
    print "Waiting for incoming connections..."
    (conn, (ip,port)) = tcpsock.accept()
    print 'Got connection from ', (ip,port)
    newthread = ClientThread(ip,port,conn)
    newthread.start()
    threads.append(newthread)

for t in threads:
    t.join()


# client3.py on local machine
#!/usr/bin/env python

#!/usr/bin/env python

import socket
import time

#TCP_IP = 'localhost'
TCP_IP = 'ip-ec2-instance'
TCP_PORT = 60001

BUFFER_SIZE = 1024
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))

clock_start = time.clock()
time_start = time.time()

with open('received_file', 'wb') as f:
    print 'file opened'
    while True:
        #print('receiving data...')
        data = s.recv(1024)
        #print('data=%s', (data))
        if not data:
            f.close()
            print 'file close()'
            break
        # write data to a file
        f.write(data)

print('Successfully get the file')
s.close()
print('connection closed')

clock_end = time.clock()
time_end = time.time()

duration_clock = clock_end - clock_start
print 'clock:  start = ',clock_start, ' end = ',clock_end
print 'clock:  duration_clock = ', duration_clock

duration_time = time_end - time_start
print 'time:  start = ',time_start, ' end = ',time_end
print 'time:  duration_time = ', duration_time
```

Server console shows the following output after a connection from my local home machine:

```
$ python server3.py
TCP_IP= ec2-...
TCP_PORT= 60001
Waiting for incoming connections...
```

```
Got connection from  ('108.239.135.40', 56742)
 New thread started for 108.239.135.40:56742
```

The ip is isp's:

Geolocation data from IP2Location (Product: DB4 updated on 6/1/2015)

IP Address	Country	Region	City	ISP
108.239.135.40	United States	California	Fair Oaks	At&t Internet Services

On my local mac:

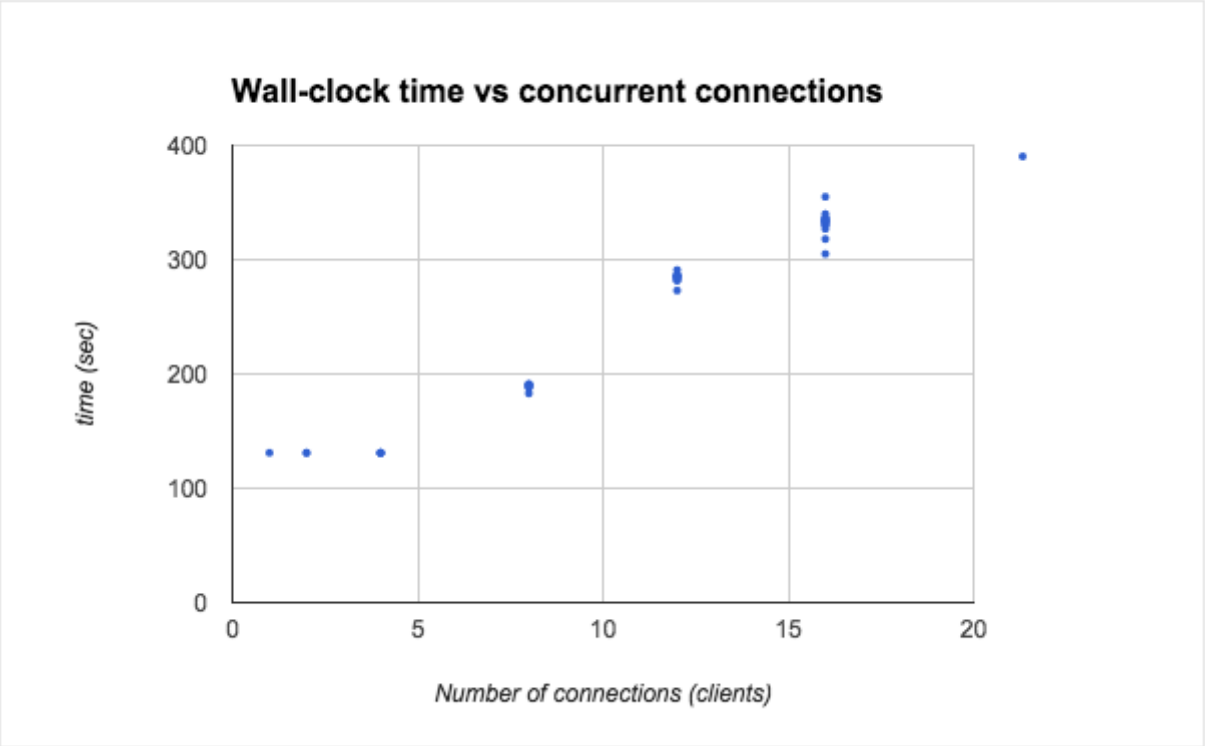
```
$ python client3.py
file opened
file close()
Successfully get the file
connection closed
clock:  start =  0.018806  end =  0.038608
clock:  duration_clock =  0.019802
time:  start =  1434991840.37  end =  1434991840.42
time:  duration_time =  0.0457620620728
```

File downloaded from EC2, **received_file** is simple, and it looks like this:

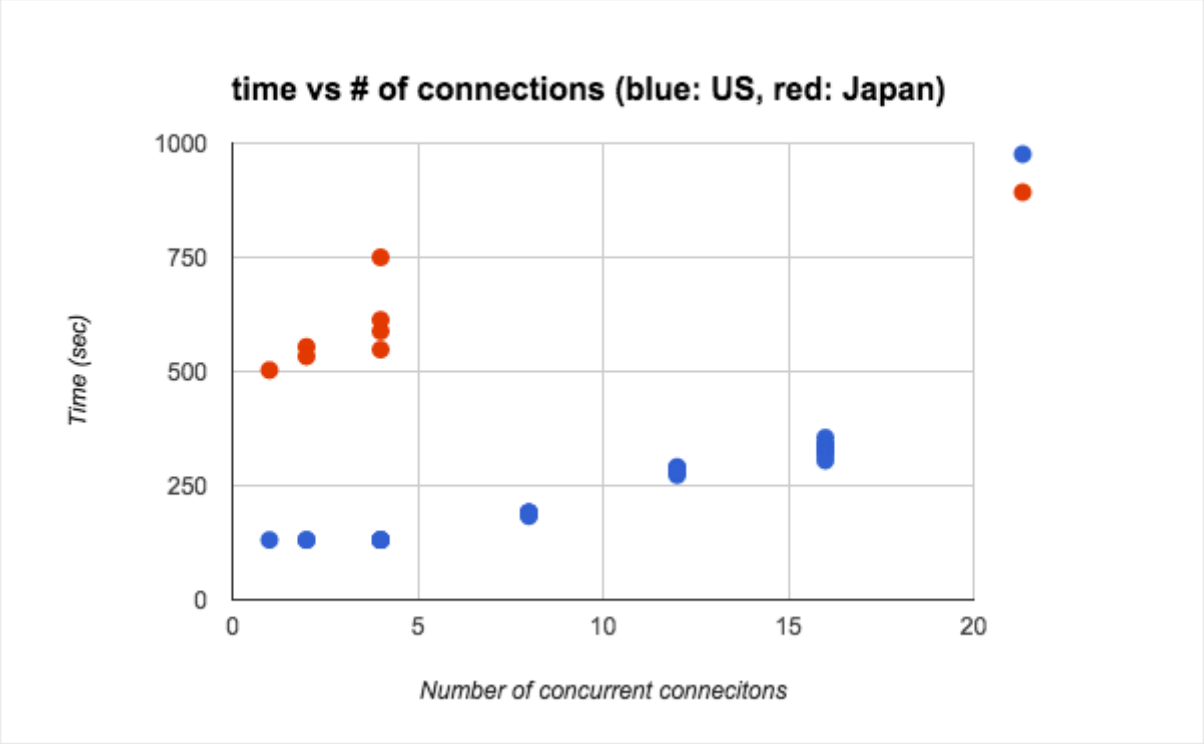
```
From EC2
1
2
3
4
5
6
7
8
9
```

Download time vs number of clients

Here is the output showing the wall-clock time depending on the number of concurrent connections:



Our server is located in California, and the following picture compares the download speed between US and Japan:



Python Network Programming

Network Programming - Server & Client A : Basics

Network Programming - Server & Client B : File Transfer

Network Programming II - Chat Server & Client

Network Programming III - SocketServer

Network Programming IV - SocketServer Asynchronous request